Fermilab

TM-1240
2311.000

EPICS SYSTEM: RSX IMPLEMENTATION ISSUES*

T. E. Lahey, J. F. Bartlett, J. S. Bobbitt, B. J. Kramper,
B. A. MacKinnon, and R. E. West

February 1984

*Submitted for publication in the Proceedings of the Digital Equipment Computer User Society, Las Vegas, Nevada, October 1983.

EPICS SYSTEM: RSX IMPLEMENTATION ISSUES

T. E. Lahey
J. Frederick Bartlett, J. S. Bobbitt, B. J. Kramper
B. A. MacKinnon, R. E. West
Fermi National Accelerator Laboratory
Batavia, Illinois

ABSTRACT

This paper presents implementation details of the Experimental Physics Interactive Control System (EPICS). EPICS is used to control accelerated particle beams for high-energy physics experiments at the Fermi National Accelerator Laboratory. The topics discussed are: interprocessor communication, support of beamline terminals and devices, resource management, mapping, various problems, some solutions to the problems, performance measurement, and modifications and extensions to RSX-11M.

This paper is the third of three related papers on the EPICS system. The other two cover (1) the system overview[1] and (2) the system structure and user interface.[2]

INTRODUCTION

EPICS is implemented with RSX11M running on PDP-11's.

The hardware configuration (figure 1) includes two closely-coupled PDP-11's: an 11/44 and an 11/34. The 11/44 is a level-2 computer that runs the user command language, utilities, and the device database. The 11/34 is a level-3 computer that reads and writes beamline devices and creates an interrupt environment for the level-2 computer. The computers are coupled with shared memory and a DR11-C interrupt link. The shared memory is located on a shared UNIBUS. Also located on this shared UNIBUS are the serial CAMAC controller and TIMER controller.

The serial CAMAC controller is a special-purpose controller providing access to serial CAMAC. The CAMAC system runs throughout the experimental areas and supports modules for interfacing to beamline devices, terminals, and experimenter computers. The CAMAC system does not provide asynchronous demands, so the level-3 polls all CAMAC modules. The serial CAMAC controller reads commands from and stores data in shared memory.

The TIMER is a second special-purpose controller that synchronizes the level-2 and level-3 computers to the external accelerator clock. The accelerator clock generates timing pulses that announce events, such as the start of an accelerator cycle and delivery of beam to the experiments. The TIMER resides on the primary UNIBUS of the level-3 computer. The CAMAC controller and TIMER are level-4 computers.

Currently, the level-2 computer runs RSX-11M and the level-3 computer runs a stripped-down version of RSX-11M that is equivalent to RSX-11S. A separate diagnostic system for the EPICS hardware runs RSX-11S on both computers. The level-3 computer has no disk. It is downloaded from the level-2 computer by using the DR11-C, shared memory, and the 11/34 boot ROM.

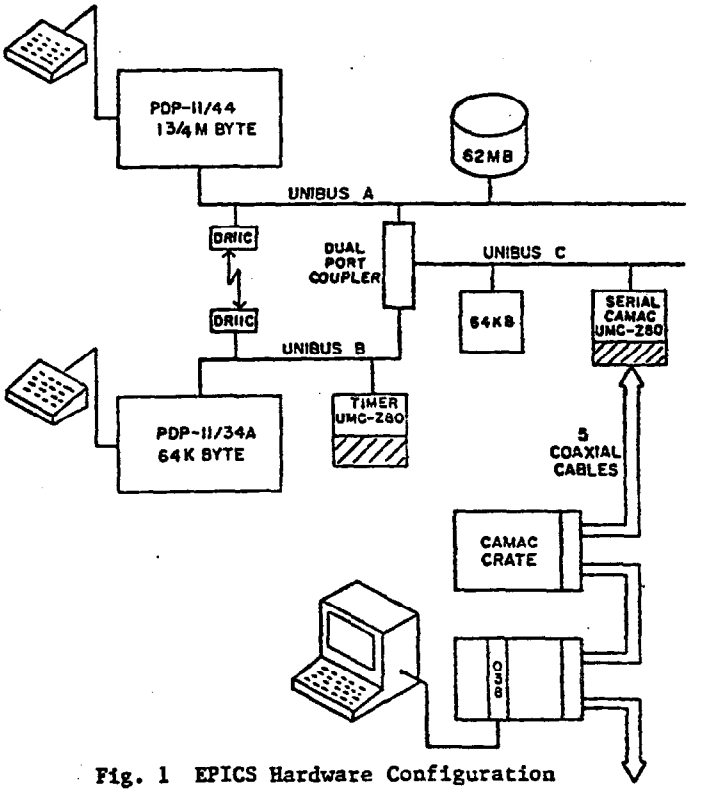

Fig. 1 EPICS Hardware Configuration

INTERPROCESSOR COMMUNICATION

The DR11-C and shared memory are used for interprocessor communication. There are three partitions in shared memory: one owned by the level-2 computer, one owned by the level-3 computer, and one containing communication queues. The initiating task creates a message in shared memory, and sends it to another task via a communication queue. The initiating task enters fork state and calls a communication routine to enqueue the message on a communication queue. When the receiving task is on the same processor, the communication handler enqueues the message on the proper queue. When the receiving task is on the other processor, the communication handler sends the address of the message, via the DR11-C, to the other processor. The communication handler on the target processor enqueues the message on the proper queue. If a task owns the communication queue, the communication handler informs the task about the arrival of a message by setting an event flag or issuing an Asynchronous System Trap (AST). The receiving task calls a routine to dequeue the message. The DR11-C A and B interrupts are supported with connect to interrupt routines (CINT$) in an RSX task.

BEAMLINE TERMINALS SUPPORT

Figure 2 shows the path through the system for access to a terminal. The user command language (CBASIC) and the utility tasks issue QIO's that are supported with a minimal driver and the Terminal Ancillary Control Processor (ACP). The ACP communicates with the level-3 terminal handler via shared-memory messages and communication queues. These messages are input and output lines. The level-3 terminal handler performs low-level processing, e.g., local-echoing and intra-line editing, and schedules polling of the terminals that are physically connected via serial CAMAC.

BEAMLINE DEVICE SUPPORT

Figure 3 shows the path through the system for access to beamline devices. The utility tasks and CBASIC issue QIO's that are supported with a minimal driver and the Request Formatter ACP. The Request Formatter communicates with the Disk Database Access task, and uses the memory-resident device database to perform initial processing on the device access request. The Request Formatter sends the request to the level-3 Request Handler via communication queues and shared memory. This task converts the request into a command list and sends the command list to the serial CAMAC driver for execution on the serial CAMAC system. Upon completion, the level-3 computer processes the data and sends it up to the utility task via the Request Formatter.

Utilities

Utilities are nonprivileged RSX tasks that typically execute QIO's to create a device access request, transfer a data buffer, and delete the device access request. An exception to this is the Page utility which is a nonprivileged task that repetitively updates many device readings on multiple terminals. There can be a maximum of 8 page displays with 45 device readings each. Page updates up to 24 of these readings every second, optionally updates another 24 devices readings every 0.2 seconds, and modifies the page displays in response to user commands.

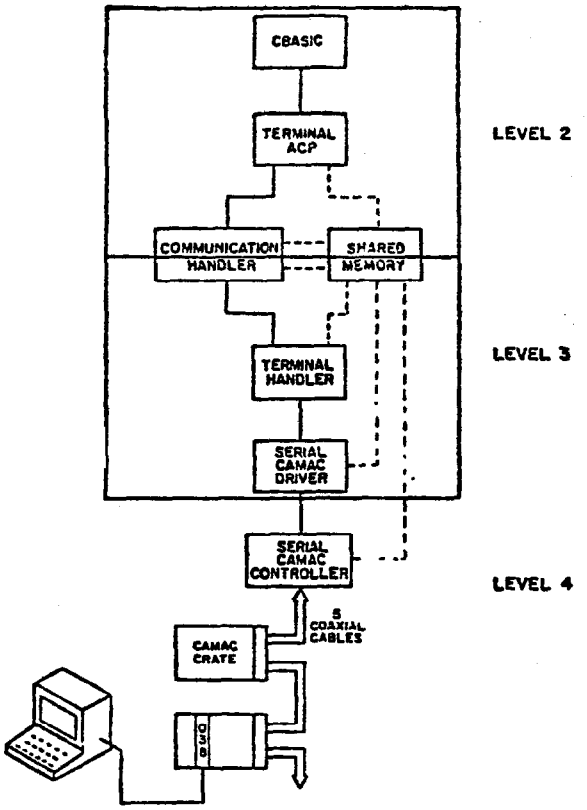

Fig. 2 Support for Beamline Terminals

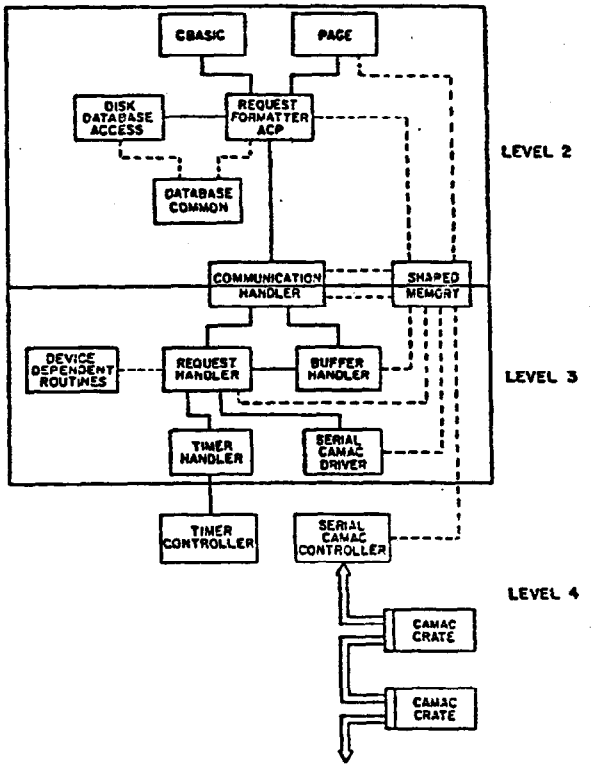

Fig. 3 Support for Beamline Devices

its device record is placed at the end of the LRU list. On the next access to the device, the Request Formatter hashes the device record, finds it on the LRU list, removes it, and starts using the device record without invoking DDA. When new device records are being loaded, DDA retrieves the least-recently-used record from the front of the LRU list and uses it for the new device. All devices that are no longer being accessed are thus eventually removed from the database common.

Another area of resource management is sharing of data when two or more read requests are identical with respect to the device, attribute, and time. The EPICS system uses a data structure in shared memory called the shared data point. The level-3 computer stores the data in the shared data point where the data is accessible by all processes that get data directly from shared memory. For example, the Page utility retrieves the same data for all users which are displaying the device. In addition, the level-3 computer distributes this data to all buffers for utilities that are retrieving the data in buffers. This mechanism gives us fewer data structures in shared memory, fewer accesses to the device, and fewer executions of routines that process that data.

Resource management is applied to beamline terminal support. On the level-3 computer, there are two polling rates: slow and fast. When no one is using a terminal, it is polled at the slow rate of once per second. When a user types the first character, the level-3 computer starts polling the terminal at the fast poll rate of 10 times per second. There is a timeout of five minutes applied to fast polling of a terminal. If a user has not typed within this time period, the level-3 computer polls the terminal at the slow polling interval.

Resource management is implemented via the concepts of defined terminals and active terminals. A defined terminal is one which has been identified to the EPICS system and which is polled by the level-3 computer. An active terminal is a terminal known to RSX through an assigned Unit Control Block (UCB). When a user types at a defined terminal, the Terminal ACP attempts to assign the terminal to a terminal UCB, making it an active terminal. This technique requires fewer UCB's in pool and hence saves pool. There is also a timeout of active terminals. If there is no activity on an active terminal, it is disconnected from the UCB, so that the active terminal port can be used for another defined terminal. This timeout and the number of active terminals allows us to support the maximum number of concurrent users without excessive use of pool. At the current time there are 24 defined terminals, 10 active terminals, and a timeout of 15 minutes on active terminals.

The level-2 utilities implement resource management via timeouts and resource quotas. For example, the Database Editor cancels an editing session if a user has typed no commands within its timeout period. Page cancels a page display if it receives no commands from the user within its timeout period. The Watch utility applies quotas on user requests. Each type of user is given a maximum number of Watch resources that can be used. For example, an active experiment can watch more devices than an inactive one.

MAPPING

All tasks that access the database common and shared memory must dynamically map to those area Currently, the database common is 16K words a shared memory is 44K words.

The Disk Database Access task is a nonprivileged task that maps to the database common via RSX PLAS directives.

To decrease the mapping time in many other tasks, the tasks map directly. The tasks modify the contents of the mapping APR's on the I/O page. For a task to directly modify its mapping under RSX, you must disable context switching. At the next context switch, RSX will recalculate the values of the mapping registers for the task. One simple solution is to disable context switching by modifying the variable SCXDBL. When nonzero, the executive will not switch to running a different task. So the task disables context switching, remaps and processes, and then reenables context switching. The next time that RSX switches the task out and back in, mapping will be as described in the window blocks.

We are starting to use another method for remapping a task. When building the task, we allocate additional window blocks with standard RSX Task Builder commands. While the task is running, it modifies the contents of its window blocks to dynamically remap. This allows fast remapping and task switching while the task executes the section of code in which it modifies mapping.

PROBLEMS

Many of the technical problems that we encountered were related to mapping. There were also some problems with RSX pool. A minor problem was seen with incorrect use of SWSTKS.

When a task disables context switching for the purpose of modifying the mapping, the task cannot issue any RSX directive that leads to a context switch. This caused minor problems related to normal use of such directives. This caused major problems related to use of the ODT debugging tool since it issues such directives.

Another problem with disabling context switching is that a task can monopolize the CPU. Additionally, you reduce your ability to tune the system with standard RSX tuning parameters, such as task priority.

If you directly modify the mapping registers, never modify APR 0. It is mapped to your task header. At minimum, the Directive Status Word (DSW) for your task is stored in your task header. The system continues to update the location at the offset of the DSW into APR 0.

As with many RSX-11M applications, we had typical problems with pool. Great care was required to achieve the largest possible pool. Otherwise there was not enough pool, especially on the level-2 computer. Additionally, when the amount of free pool was low, it became difficult or impossible to diagnose or solve the problems.

One minor problem was the use of a SWSTKS from an illegal APR. In a 20K executive, RSX processes a SWSTKS by copying the user mapping registers 5 to 7

to the corresponding kernal mapping regisers. The executive then transfers control to the code following the SWSTK$. If this code is in APR 0 through 4, the executive is not mapped to your code, and it will execute whatever is at that location in the kernal address space.

SOLUTIONS

There are multiple solutions to the above problems, some of which first became available during the development of EPICS.

Conversion from RSX-11M to RSX-11M+ on the level-2 computer may solve many problems, such as those with pool. We will use multi-user tasks to reduce the number of tasks and hence the amount of task swapping. Use of Instruction and Data Space gives a larger task address space and can reduce the amount of dynamic mapping.

For dynamic mapping of a task's address space, we are increasing the usage of window blocks allocated by the Task Builder and modified directly at execution time. Another possible solution to some of our mapping problems is to use a VAX.

To use ODT for debugging a task that disables context switching while it remaps, we added the ability to preserve the mapping of one task across task switches. When debugging a task, the task identifies itself as the task for which mapping is preserved, and then remaps without disabling context switching.

We can move the data structures associated with the Request Formatter ACP from RSX pool to M+ secondary pool or to another partition. Additionally, we are moving all data structures that need not reside in shared memory to other locations, thus using shared memory only when required. More effective use of this critical resource allows us to reduce allocation restrictions and to extend the automatic recovery timeouts.

For the communication handler and TIMER handler, we are converting to RSX drivers from the connect-to-interrupt mechanism. This reduces the time to handle an interrupt. Additionally, this removes the context switch which occurs when these devices are supported with an RSX task. The privileged tasks will enter fork state and directly queue commands to the drivers via the routine $DRQRQ. Future code can be nonprivileged and access the drivers via QIO's.

Newer versions of the compiler have features that allow us to write more code in PASCAL. This provides for faster implementation and more maintainable code. We currently have a stripped-down PASCAL run-time system that we will use to write privileged code in PASCAL. For example, new versions of the Terminal ACP will be coded in PASCAL.

We are considering a modification to the code which supports SWSTK$, such that a PR:0 task (privileged without being mapped to the executive) can execute a SWSTK$ to enter fork state, provided the SWSTK$ code is located in APR 5 or 6 for a 20K executive. We will also check whether the task has executed the SWSTK$ from a legal APR.

An improved method for support of critical sections between the Disk Database Access task and Request Formatter is the use of global event flags. The task priorities can then be set according to relative processing priority of the tasks.

We are migrating the low-level terminal processing from the level-3 computer to intelligent terminals. This reduces the level-3 computer load and provides better response to a user typing at the terminal. In many cases, these terminals can be supported by a local (level-5) computer, e.g., personal computer. This method provides even more power to the user and off-loads more functions from the level-2 and level-3 computers.

PERFORMANCE MEASUREMENT

To increase the real-time response of the EPICS system, we measured performance in two ways.

The first method involves attaching a logic state analyzer to the UNIBUS address lines. By attaching the analyzer to the primary UNIBUS of the 11/44 or 11/34, we can observe relative execution of code on that processor. By attaching the analyzer to the shared UNIBUS, we can observe overall use of the shared memory by the 11/44, 11/34, and serial CAMAC controller.

A second method involves use of a DR11-C attached to an oscilloscope via Fermilab equipment that conditions the DR11-C signals. An EPICS task or driver sets and clears individual bits in the DR11-C register to indicate its execution and its states. The oscilloscope generates a trace of the reported activity. We thus get a more detailed idea of system activity and timing.

MODIFICATIONS AND EXTENSIONS OF RSX

To produce the implementation of EPICS that we have described, we made minimal modifications and extensions to the executive.

Currently there are two modifications to the system. First, the task switching code was changed to preserve the mapping of a single task across context switches. Second, we modified XDT on the level-3 computer so that the system would reboot without an operator-entered command, i.e., G command to XDT.

Additional functionality was supported by extensions to the executive. During the RSX system generation, we added routines for allocation and deallocation of shared memory and for interprocessor communication. At system reboot, the RSX illegal instruction code is extended by loading code into pool and changing the executive code to branch to the new code. We have programs to load and remove the illegal instruction code whenever necessary.

As mentioned earlier, we will be modifying the SWSTK$ code to allow its use from a privileged task that is not mapped to the executive.

CONCLUSIONS

Many of the implementation techniques, such as the use of ACP's, special-purpose drivers, and fast execution of code are applicable to both RSX-11M and to RSX-11M+. This discussion is useful for time-critical RSX-11M applications on 18-bit machines. Some of the methods we used in implementing the EPICS control system can be avoided by use of RSX-11M+.

REFERENCES

1. Bartlett, J.F., et al., "The EPICS System: An Overview", DECUS Proceedings, Fall 1983.
2. West, R.E., et al., "EPICS System: System Structure and User Interface", DECUS Proceedings, Fall 1983.